BCIT

COMP 7005 - Computer Network and Protocols

Data Communications Principles

Final Project: Protocol Design

A01059056 - Alex Laroche

A00953335 - Kalen Tara

Introduction:

The main purpose of this project is to design and implement a **Send-And-Wait** protocol. The protocol will be based around the half duplex system while also using sliding windows. This system will send multiple packets between the host and the client on a LAN network with unreliable connections between the server and the client. Our design requires us to discard random packets to mimic the noise of an unreliable network.

Essentially the project can be broken into three devices. A transmitter, a receiver and a noise generating unreliable network emulator. This unreliable network emulator will mirror a certain bit error rate. This will be done using three physical machines over a LAN connection.

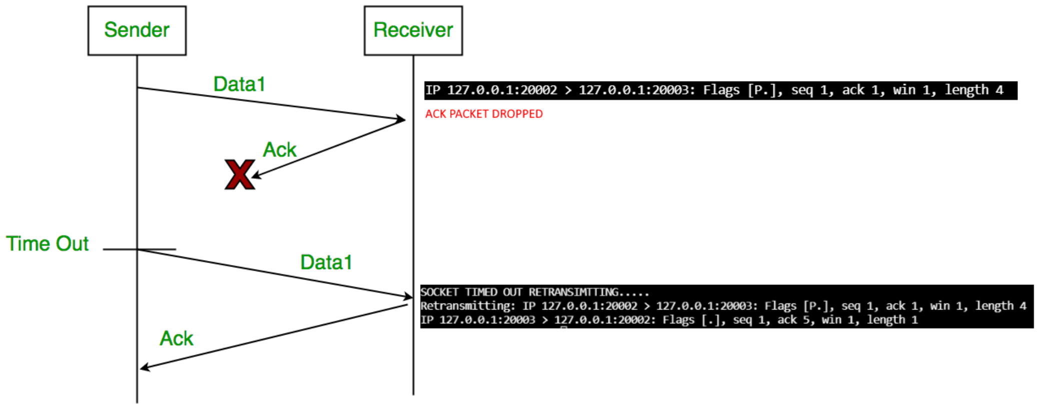
Our design is based on taking a UDP connection and converting it into a connection that acts like a TCP connection. This paper will walk you through our design and the implementation that we followed.

Primarily the protocol created is primarily focused on the transmitter with it sending the data through the noise-simulated network and reaching the receiver which sends back ACKs to the sent data accordingly.

Design:

Basic Timeout Design:

The first step was ensuring the understanding of the Send-And-Wait protocol or better known as the Stop-And-Wait protocol. Where the Transmitter(Sender) would send data to the Receiver. The Transmitter would then wait for the ACK correlated to the sent packet. If the ACK does not return within the specific timeout time, The data will be resent to the server, this will continue until the data has been sent and the ACK for the data has been received. If the network drops a data packet or an ACK, then the transmitter will wait until the specified timeout counter has been reached (located in the config) and retransmits all the packets which have been sent in the window.

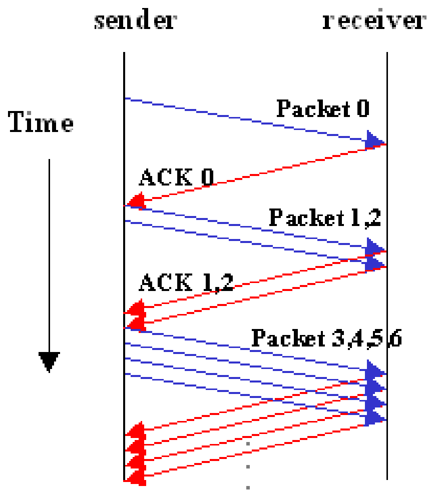


```
IP 127.0.0.1:20002 > 127.0.0.1:20003: Flags [P.], seq 1, ack 1, win 1, length 4
ACK PACKET DROPPED
```

```
SOCKET TIMED OUT RETRANSIMTTING.....
Retransmitting: IP 127.0.0.1:20002 > 127.0.0.1:20003: Flags [P.], seq 1, ack 1, win 1, length 4
IP 127.0.0.1:20003 > 127.0.0.1:20002: Flags [.], seq 1, ack 5, win 1, length 1
```

Sliding window:

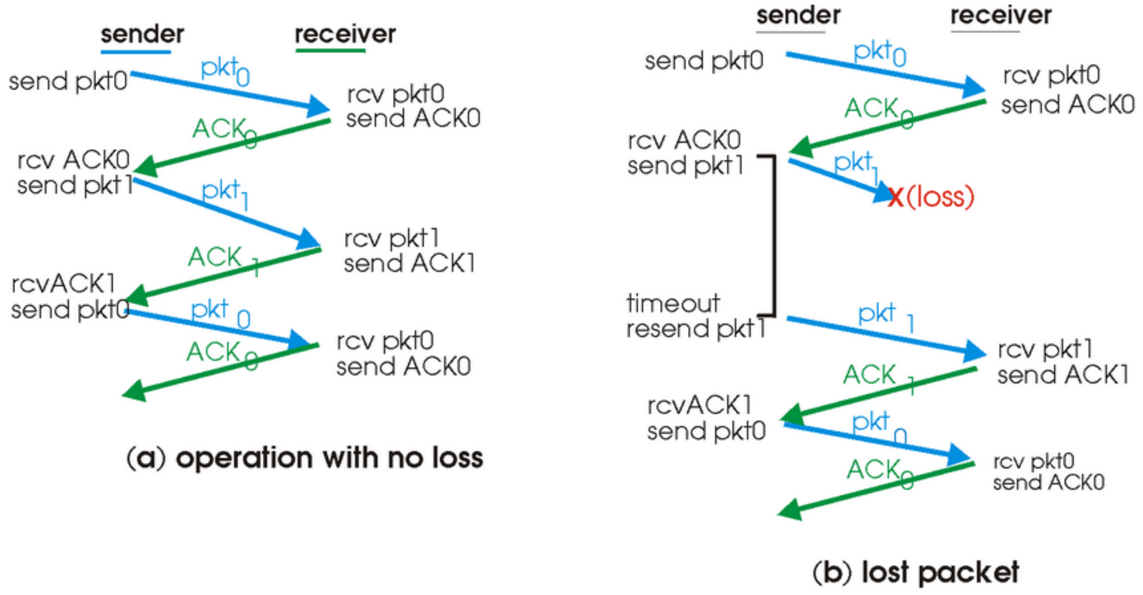
The next step was designing a sliding window which would slowly increase until it reached its largest specified size. The primary use of a sliding window is to establish a form of flow control of data in transit on a network. It also allows for easier retransmission as the transmitter would only have to retransmit the data from the specific window segment as opposed to the whole data. With each ACK received, the Window Size increases by a multiple of 2 until it reaches the maximum window size specified in the transmitter config. The subsequent data packets will not be sent until all the previous data packets have been received. Which means the Window size will not increase until the data packets have been properly sent and received.

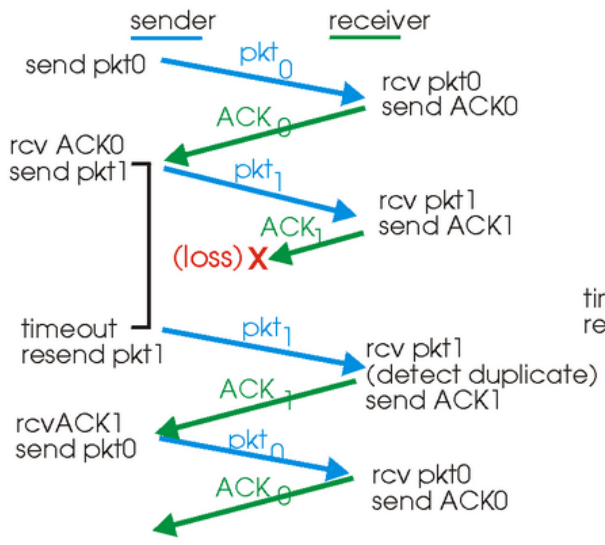


RDT 3.0 protocol:

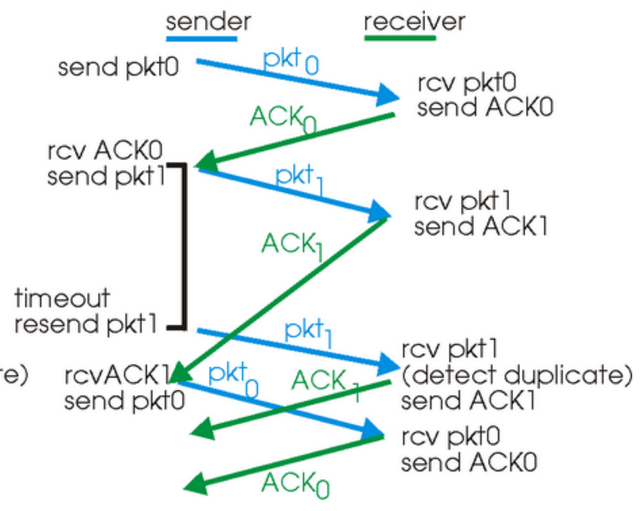
"From the sender's viewpoint, retransmission is a panacea. The sender does not know whether a data packet was lost, an ACK was lost, or if the packet or ACK was simply overly delayed. In all cases, the action is the same: retransmit. In order to implement a time-based retransmission mechanism, a countdown timer will be needed that can interrupt the sender after a given amount of timer has expired. The sender will thus need to be able to *(i)* start the timer each time a packet (either a first time packet, or a retransmission) is sent, *(ii)* respond to a timer interrupt (taking appropriate actions), and *(iii)* stop the timer."(http://www2.ic.uff.br/~michael/kr1999/3-transport/3_040-principles_rdt.htm)

In all instances of the transmitter not receiving the ACK within the specific time frame, then the transmitter will retransmit the packet data. The timer will reset with every subsequent ACK received, until all the ACKs have been received then the transmitter will end the next data packet.





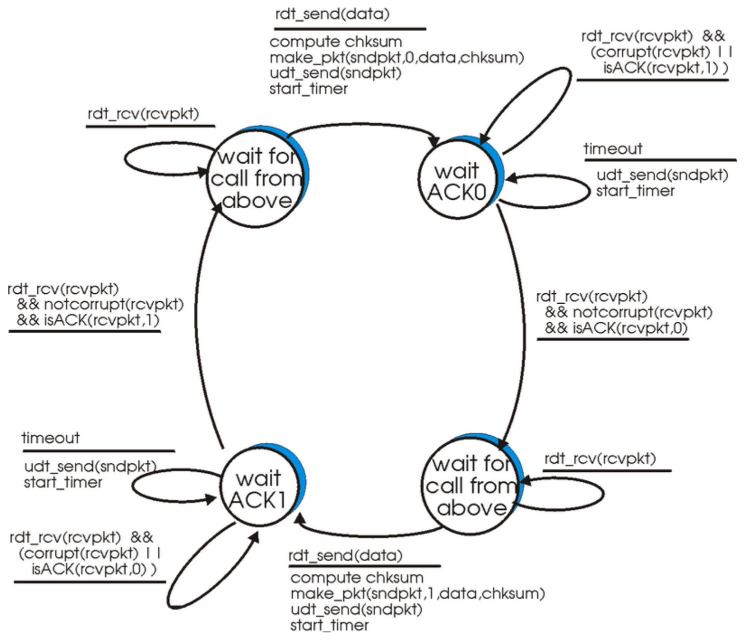
(c) lost ACK



(d) premature timeout

Transmitter

The transmitter will loop through this sequence until all the data has been sent and all the ACKs have been received. 1. Send the packet, start timer 2. If the data is received continue, else timeout, resend the data and restart the timer. 3. Receive ACK and send the next data.


rdt_send(data)
compute chksum
make_pkt(sndpkt,0,data,chksum)
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))
rdt_rcv(rcvpkt)
wait for call from above
wait ACK0
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)
timeout
udt_send(sndpkt)
start_timer
wait ACK1
wait for call from above
rdt_rcv(rcvpkt)
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,0))
rdt_send(data)
compute chksum
make_pkt(sndpkt,1,data,chksum)
udt_send(sndpkt)
start_timer

Conclusion:

In conclusion the protocol created was a combination of the rdt 3.0 and sliding window with the beginning of slow start as the window size increased with a multiple of 2 till it reached the maximum specified window size. The window size throughout the transaction is not dependent on the lost packets and retransmissions it reaches and remains at the maximum window size until the transmission is completed. The planning and implementation of this protocol was too challenging, however understanding how Sequence Numbers and Acknowledgement Numbers interact with each other and how to increment them properly.